华南理工大学

South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
邢浩、何嘉豪、黄迦密

Supervisor:
Mingkui Tan

Student ID：201530613221（邢浩）201530361276（何嘉豪）
201537611718（黄迦密）

Grade:
Undergraduate

December 9, 2017

Face Classification Based on AdaBoost Algorithm

Abstract—

I. INTRODUCTION

A. Motivation of Experiment

1. Understand Adaboost further
2. Get familiar with the basic method of face detection
3. Learn to use Adaboost to solve the face classification problem,and combine the theory with the actual project
4. Experience the complete process of machine learning

B. Dataset

1. This experiment provides 1000 pictures, of which 500 are human face RGB images, stored in *datasets/original/face*; the other 500 is a non-face RGB images, stored in *datasets/original/nonface*.
2. The dataset is included in the example repository. Please download it and divide it into training set and validation set.

C. Environment for Experiment

python3, at least including following python package: sklearn，numpy, matplotlib, pickle, PIL.
It is recommended to install anaconda3 directly, which has built-in python package above.
PyCharm Community Integrated Development Environment (optional)

II. METHODS AND THEORY

AdaBoost is an iterative algorithm. The core idea of AdaBoost is to train different classifiers, ie weak classifiers, against the same training set, and then combine these weak classifiers to construct a stronger final classifier. The algorithm itself is to change the distribution of data to determine the weight of each sample based on the correct classification of each sample in each training set and the accuracy of the last overall classification. The new data of the modified weights are sent to the lower classifier for training, and then the classifiers obtained by each training are fused together to be the final decision classifier

III. EXPERIMENT

A. Experiment Step

1. Read data set data. The images are supposed to converted into a size of 24 * 24 grayscale, the number and the proportion of the positive and negative samples is not limited, the data set label is not limited.
2. Processing data set data to extract NPD features. Extract features using the NPDFeature class in feature.py. (Tip: Because the time of the pretreatment is relatively long, it can be pretreated with pickle function library dump () save the data in the cache, then may be used load () function reads the characteristic data from cache.)
3. The data set is divisded into training set and calidation set, this experiment does not divide the test set.
4. Write all *AdaboostClassifier* functions based on the reserved interface in *ensemble.py*. The following is the guide of *fit* function in the *AdaboostClassifier* class:
   4.1 Initialize training set weights w, each training sample is given the same weight.
   4.2Training a base classifier , which can be sklearn.tree library DecisionTreeClassifier (note that the training time you need to pass the weight w as a parameter).
   4.3 Calculate the classification error rate w of the base classifier on the training set.
   4.4 Calculate the parameter w according to the classification error rate w .
   4.5 Update training set weights w
   4.6 Repeat steps 4.2-4.6 above for iteration, the number of iterations is based on the number of classifiers.
5. Predict and verify the accuracy on the validation set using the method in AdaboostClassifier and use classification_report () of the sklearn.metrics library function writes predicted result to *report.txt* .

B. Code

feature.py

```
import numpy


class NPDFeature():
    """It is a tool class to extract the NPD features.

    Attributes:
        image: A two-dimension ndarray indicating grayscale image.
        n_pixels: An integer indicating the number of image total pixels.
        features: A one-dimension ndarray to store the extracted NPD features.
    """
    __NPD_table__ = None

    def __init__(self, image):
        '''Initialize NPDFeature class with an image.'''
        if NPDFeature.__NPD_table__ is None:
```

```
        NPDFeature.__NPD_table__ = NPDFeature.__calculate_NPD_table()
        assert isinstance(image, numpy.ndarray)
        self.image = image.ravel()
        self.n_pixels = image.size
        self.features = numpy.empty(shape=self.n_pixels * (self.n_pixels - 1) // 2, dtype=float)

    def extract(self):
        '''Extract features from given image.

        Returns:
            A one-dimension ndarray to store the extracted NPD features.
        '''
        count = 0
        for i in range(self.n_pixels - 1):
            for j in range(i + 1, self.n_pixels, 1):
                self.features[count] = NPDFeature.__NPD_table__[self.image[i]][self.image[j]]
                count += 1
        return self.features

    @staticmethod
    def __calculate_NPD_table():
        '''Calculate all situations table to accelerate feature extracting.'''
        print("Calculating the NPD table...")
        table = numpy.empty(shape=(1 << 8, 1 << 8), dtype=float)
        for i in range(1 << 8):
            for j in range(1 << 8):
                if i == 0 and j == 0:
                    table[i][j] = 0
                else:
                    table[i][j] = (i - j) / (i + j)
        return table
```

ensemble.py

```
import pickle
import numpy as np
import math

class AdaBoostClassifier:
    '''A simple AdaBoost Classifier.'''

    def __init__(self, weak_classifier, n_weakers_limit):
        '''Initialize AdaBoostClassifier

        Args:
            weak_classifier: The class of weak classifier, which is recommend to be sklearn.tree.DecisionTreeClassifier.
            n_weakers_limit: The maximum number of weak classifier the model can use.
        '''
        self.weak_classifier = weak_classifier
        self.n_weakers_limit = n_weakers_limit
        pass

    def is_good_enough(self):
        '''Optional'''
        pass

    def fit(self,X,y):
        '''Build a boosted classifier from the training set (X, y).

        Returns:
            X: An ndarray indicating the samples to be trained, which shape should be (n_samples,n_features).
            y: An ndarray indicating the ground-truth labels correspond to X, which shape should be (n_samples,1).
        '''
        trees = []
        effects = []
        num_samples, num_features = np.shape(X)
        weights = np.ones(num_samples)/num_samples
        class_dist = np.zeros(num_samples)
        for i in range(self.n_weakers_limit):
            clf = self.weak_classifier
            best_tree = clf.fit(X, y, weights)
            y_pre = clf.predict(X)
            precision = np.mean((y == y_pre))
            error = 1-precision
            alpha = 0.5*math.log(1/(error+1e-8)-1)
            trees.append(best_tree)
            effects.append(alpha)
            exp_factor = -alpha * y * y_pre
            weights = weights * np.exp(exp_factor)
            weights = weights / weights.sum()
            class_dist += alpha * y_pre
            sum_precision = np.mean((y == np.sign(class_dist)))
            print(i+1, precision, sum_precision)
        self.save(trees, './trees')
        self.save(effects, './effects')
        pass

    def predict_scores(self, X):
        '''Calculate the weighted sum score of the whole base classifiers for given samples.

        Args:
            X: An ndarray indicating the samples to be predicted, which shape should be (n_samples,n_features).

        Returns:
            An one-dimension ndarray indicating the scores of differnt samples, which shape should be (n_samples,1).
        '''
        s, f = np.shape(X)
        scores = np.zeros(s)
        trees = self.load('./trees')
        effects = self.load('./effects')
        for i in range(self.n_weakers_limit):
```

```
            tree = trees[i]
            v = tree.predict(X)
            scores += v * effects[i]
        return scores
        pass

    def predict(self, X, threshold=0):
        '''Predict the catagories for geven samples.

        Args:
            X: An ndarray indicating the samples to be predicted, which shape should be (n_samples,n_features).
            threshold: The demarcation number of deviding the samples into two parts.

        Returns:
            An ndarray consists of predicted labels, which shape should be (n_samples,1).
        '''
        scores = self.predict_scores(X)
        labels = np.sign(scores-threshold)
        return labels
        pass

    @staticmethod
    def save(model, filename):
        with open(filename, "wb") as f:
            pickle.dump(model, f)

    @staticmethod
    def load(filename):
        with open(filename, "rb") as f:
            return pickle.load(f)
```

train.py

```
from ensemble import AdaBoostClassifier
from feature import NPDFeature
import os
from PIL import Image
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
import pickle


def pre_process(dir, file):
    features = np.array([])
    for filename in os.listdir(dir):
        img = Image.open(os.path.join(dir, filename))
        resize_img = img.resize((24, 24))
        gray_img = np.array(resize_img.convert("L"))
        feature = NPDFeature(gray_img).extract()
        features = np.append(features, feature).reshape(-1, 165600)
        print(features.shape)
    pass
    with open(file, "wb") as f:
        pickle.dump(features, f)


pre_process('./datasets/original/face', "face.npy")
pre_process('./datasets/original/nonface', "nonface.npy")

if __name__ == "__main__":
    # write your code here
    X1 = pickle.load(open("face.npy", "rb"))
    y1 = np.ones(500)
    X2 = pickle.load(open("nonface.npy", "rb"))
    y2 = -1 * np.ones(500)
    X = np.append(X1, X2).reshape(-1, 165600)
    y = np.append(y1, y2)

    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)

    clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=6), 10)
    clf.fit(X_train, y_train)
    y_pre = clf.predict(X_val)
    accuracy = np.mean((y_val == y_pre))
    print("acc = ", accuracy)
    report = classification_report(y_val, y_pre, labels=[1, -1],\
                        target_names=['face', 'nonface'],\
                        digits=4)
    with open('./report.txt', 'w') as f:
        f.write(report)
    pass
```

Result

	precision	recall	f1-score	support
face	0.8922	0.8750	0.8835	104
nonface	0.8673	0.8854	0.8763	96
avg / total	0.8802	0.8800	0.8800	200

IV. CONCLUSION

Through this experiment, I was further acquainted with the principle of adaboost, learned to deal with image features, a very meaningful experiment